Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 398 (2008) 114–128
Theoretical Computer Science
www.elsevier.com/locate/tcs

Characterizing strong normalization in the Curien–Herbelin symmetric lambda calculus: Extending the Coppo–Dezani heritage

Daniel J. Dougherty[a,*], Silvia Ghilezan[b], Pierre Lescanne[c]

[a] *Worcester Polytechnic Institute, Worcester, MA 101609, USA*
[b] *Faculty of Engineering, University of Novi Sad, Novi Sad, Serbia*
[c] *École Normale Supérieure de Lyon, Lyon, France*

Abstract

We develop an intersection type system for the $\overline{\lambda}\mu\widetilde{\mu}$ calculus of Curien and Herbelin. This calculus provides a symmetric computational interpretation of classical sequent style logic and gives a simple account of call-by-name and call-by-value. The present system improves upon earlier type disciplines for $\overline{\lambda}\mu\widetilde{\mu}$: in addition to characterizing the $\overline{\lambda}\mu\widetilde{\mu}$ expressions that are strongly normalizing under free (unrestricted) reduction, the system enjoys the Subject Reduction and the Subject Expansion properties.

Keywords: Intersection-types; Classical logic; Sequent calculus

1. Introduction

Intersection type assignment systems, introduced into the lambda calculus in the late 1970s by Coppo and Dezani [10,11], were devised in order to type more lambda terms than the basic simply typed system. Indeed, these intersection types systems can characterize exactly the strongly normalizing lambda terms, and are suitable for analyzing λ-models and various normalization properties of λ-terms. In this paper, we are interested in the properties of reduction in the $\overline{\lambda}\mu\widetilde{\mu}$ calculus of Curien and Herbelin [14]. The $\overline{\lambda}\mu\widetilde{\mu}$ calculus is a term calculus embodying a Curry–Howard propositions-as-types correspondence for classical logic. We define a new type system featuring intersection types which serves to characterize strong normalization in $\overline{\lambda}\mu\widetilde{\mu}$. In contrast to earlier work, including the current authors' [19,20], we characterize SN for free, or unrestricted, reduction, rather than simply the call-by-name or call-by-value subsystems.

Under the traditional Curry–Howard correspondence, formulae provable in intuitionistic logic coincide with types inhabited in the simply typed λ calculus. Griffin extended this correspondence to classical logic in his seminal 1990 POPL paper [24], by observing that classical tautologies suggest typings for certain control operators. This initiated an active line of research; in particular the $\overline{\lambda}\mu$ calculus of Parigot [33] embodies a Curry–Howard correspondence for classical logic based on natural deduction.

* Corresponding author.
E-mail addresses: dd@cs.wpi.edu (D.J. Dougherty), gsilvia@uns.ns.ac.yu (S. Ghilezan), Pierre.Lescanne@ens-lyon.fr (P. Lescanne).

0304-3975/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.01.022

Meanwhile, Curien and Herbelin [14,27], building on earlier work in [26], defined the system $\overline{\lambda}\mu\widetilde{\mu}$. In contrast to Parigot's $\overline{\lambda}\mu$-calculus, which bases its type system on a natural deduction system for classical logic, expressions in $\overline{\lambda}\mu\widetilde{\mu}$ represent derivations in a *sequent calculus* proof system and reduction reflects the process of cut-elimination. As described in [14], the sequent calculus basis for $\overline{\lambda}\mu\widetilde{\mu}$ supports an interpretation of the reduction rules of the system as operations of an abstract machine. In particular, the right- and left-hand sides of a sequent directly represent the *code* and *environment* components of the machine. This perspective is elaborated more fully in [13].

In this paper, the type-system is presented in "one-sided" sequent style: negation on types is an involution, in the sense that we identify a type with its double-negation. Note that we are still in a *sequent calculus* rather than a natural deduction system (a crucial aspect of $\overline{\lambda}\mu\widetilde{\mu}$, as emphasized by Curien and Herbelin) since our typing rules are all introductions, with no elimination rules.

A useful perspective emerges if we compare the present project to the study of types in the standard λ-calculus. Type systems have been used in order to interpret λ-terms as defining set-theoretic functions (simple types), and later to enforce data abstraction (dependent and polymorphic types). Roughly, this use of types enables the λ-calculus to be used as an *applied* calculus. But in another direction, type systems were developed to study the reduction behavior of λ-terms and the structure of models. Intersection types, introduced into the λ-calculus by Coppo and Dezani [10,11], Pottinger [37] and Sallé [40], play a central role in this analysis. Key results are the characterizations of terms that are solvable, normalizing, and strongly normalizing in terms of their possible typings [12,37,15], and the completeness results for set-theoretic semantics [4]. In a precise sense the paradigms of types-as-propositions and types for operational and denotational semantics are skew to each other; as pointed out by Hindley [28], there does not seem to be any standard logical notion that corresponds to intersection.

Related work. Curien and Herbelin [14] encode simply-typed call-by-name and call-by-value $\overline{\lambda}\mu\widetilde{\mu}$ into the simply-typed λ-calculus via CPS translations: this implies strong normalization for these reductions. Lengrand [30] and Polonovski [35] consider the question of strong normalization for simply-typed terms under free (unrestricted) reduction. The free reduction relation in $\overline{\lambda}\mu\widetilde{\mu}$ has a critical pair, a reflection of the inherent symmetry in the system, but it complicates reasoning about reduction; indeed this system is not confluent. In [30] Lengrand shows how simply-typed $\overline{\lambda}\mu\widetilde{\mu}$ and the calculus of Urban and Bierman [42] are mutually interpretable, so that the strong normalization proof of the latter calculus yields a proof of strong normalization for free simply-typed $\overline{\lambda}\mu\widetilde{\mu}$. Polonovski [35] presents a proof of SN for the simply-typed free calculus with a method based on the "symmetric candidates" idea of Barbanera and Berardi [1] (actually Polonovski treats a version of $\overline{\lambda}\mu\widetilde{\mu}$ with explicit substitutions). David and Nour [16] present an arithmetic proof of SN for Parigot's $\overline{\lambda}\mu$-calculus.

Prior to [14], several term-assignment systems for the sequent calculus were proposed as a tool for studying the process of cut-elimination [36,5,42]. In these systems – with the exception of the one in [42] – expressions do not unambiguously encode sequent derivations.

This paper is informed by our earlier work [19,20] on intersection and union types in symmetric λ calculi. These papers characterized strong normalization for call-by-name and call-by-value restrictions $\overline{\lambda}\mu\widetilde{\mu}$; the results of the present paper apply to unrestricted reduction. General consideration of symmetry (cf. the discussion on Section 3.1) led us earlier to consider union types together with intersection types in our system. It is well-known [34,2] that the presence of union types causes difficulties for the Subject Reduction property; unfortunately our attempt to recover Subject Reduction in [19] was in error, as was pointed out to us by Hugo Herbelin [25]. Details are presented in the discussion (Section 4.2).

The larger context of related research includes a wealth of work in logic and programming languages. We described above the fundamental importance of intersection types for λ-calculus. In the 1980s and early 1990s, Reynolds explored the role that intersection types can play in a practical programming language (see for example the report [38] on the language Forsythe).

The paper is organized as follows. Section 2 deals with the untyped syntax of $\overline{\lambda}\mu\widetilde{\mu}$. Section 3 presents an intersection type system $\mathcal{M}^{\cap}$. Basic structural properties of the system are investigated in Section 4. Subject reduction is proved. In Section 5 we prove the typability of normal forms in this system and the typability of strongly normalizing $\overline{\lambda}\mu\widetilde{\mu}$-expressions. In Section 6, we give a proof of strong normalization under free reduction, for expressions typable in $\mathcal{M}^{\cap}$. Finally, in Section 7 we discuss some open problems.

2. Syntax of the $\bar{\lambda}\mu\tilde{\mu}$-calculus

The untyped syntax of $\bar{\lambda}\mu\tilde{\mu}$-calculus [14] consists of three syntactic categories: *terms*, *coterms*, and *commands*. Terms yield values, while coterms consume values. A command is a cut of a term against a coterm. An *expression* is a term, a coterm, or a command.

Formally, we fix two disjoint infinite sets Var_r and Var_e of *variables* and *co-variables* respectively. In our concrete syntax, lowercase Latin letters $x, y, \ldots$ range over variables and lowercase Greek letters $\alpha, \beta, \ldots$ range over covariables. It will be convenient to use v to range over $Var_r \cup Var_e$.

The syntax of $\bar{\lambda}\mu\tilde{\mu}$ expressions is given as follows.

$$\begin{array}{lrcl} \text{Term:} & r & ::= & x \mid \lambda x.r \mid \mu\alpha\,.\,c \\ \text{Coterm:} & e & ::= & \alpha \mid r \bullet e \mid \tilde{\mu}x\,.\,c \\ \text{Command:} & c & ::= & \langle r \parallel e \rangle \end{array}$$

In $\lambda x.r$, mxc, and $\mu\alpha\,.\,c$ the indicated (co)variables are bound by λ, $\tilde{\mu}$, or μ, respectively. The sets of free and bound variables and covariables are defined as usual, respecting Barendregt's convention [3] that no variable (covariable) can be both bound and free in the expression. The set $\mathrm{Fv}(t)$ denotes the set of all free (co)variables of an expression t.

The reduction rules of the calculus are:

$$\begin{array}{lrcl} (\lambda) & \langle \lambda x.r' \parallel r \bullet e \rangle & \longrightarrow & \langle r'[x \leftarrow r] \parallel e \rangle \\ (\mu\text{-red}) & \langle \mu\alpha.c \parallel e \rangle & \longrightarrow & c[\alpha \leftarrow e] \\ (\tilde{\mu}\text{-red}) & \langle r \parallel \tilde{\mu}x.c \rangle & \longrightarrow & c[x \leftarrow r] \end{array}$$

Of course, the substitutions above are defined so as to avoid variable capture. The reflexive and transitive closure of the reduction relation will be denoted by $\xrightarrow{*}$.

As a rewriting calculus $\bar{\lambda}\mu\tilde{\mu}$ has an essential critical pair between the μ and the $\tilde{\mu}$ redexes. That is to say, in an expression of the form $\langle \mu\alpha.c \parallel \tilde{\mu}x.c \rangle$ rules (μ) and $(\tilde{\mu})$ can be applied ambiguously. As Curien and Herbelin [14] observe:

- if one gives priority to (μ-red) over ($\tilde{\mu}$-red), this corresponds to a *call-by-value* discipline while
- if one gives priority to ($\tilde{\mu}$-red) over (μ-red), this corresponds to a *call-by-name* discipline.

Wadler [45,46] also stresses this identification in his Dual Calculus, a system closely related to $\bar{\lambda}\mu\tilde{\mu}$. Indeed the calculus is inherently not confluent. As a simple example, observe that the command $\langle \mu\alpha.\langle y \parallel \beta \rangle \parallel \tilde{\mu}x.\langle z \parallel \gamma \rangle \rangle$ reduces to each of $\langle y \parallel \beta \rangle$ and $\langle z \parallel \gamma \rangle$.

This is more than simply a reflection of the well-known fact that the equational theories of call-by-name and call-by-value differ. It is a reflection of the great expressive power of the language: a single expression containing several commands can encompass several complete computational processes, and the μ and $\tilde{\mu}$ reductions allow free transfer of control between them.

So the combinatorics of pure reduction are very complex. In this light, it is perhaps slightly surprising that the strongly normalizing computations can so readily be characterized, via the type system we present later.

When reduction in $\bar{\lambda}\mu\tilde{\mu}$ is constrained to commit to the call-by-name discipline or to the call-by-value one, the system is confluent. Confluence of $\bar{\lambda}\mu\tilde{\mu}$ and of the Dual Calculus introduced by Wadler [45,46] has been proven in [31] and [21].

It is not hard to see that pure $\bar{\lambda}\mu\tilde{\mu}$ is Turing-complete as a programming language, since the untyped λ-calculus can be coded easily into it. Space does not permit a formal development here.

The following observation, analogous to the "promotion of head-reductions" techniques from the λ-calculus, is straightforward but will be extremely useful in the sequel.

Lemma 1 (*Promotion of Top-Reductions*).

- *If* $\langle \lambda x.r_1 \parallel r \bullet e \rangle \xrightarrow{*} \langle \lambda x.r_1' \parallel r' \bullet e' \rangle \longrightarrow \langle r_1'[x \leftarrow r'] \parallel e' \rangle$
  then $\langle \lambda x.r_1 \parallel r \bullet e \rangle \longrightarrow \langle r_1[x \leftarrow r] \parallel e \rangle \xrightarrow{*} \langle r_1[x \leftarrow r'] \parallel e' \rangle$.
- *If* $\langle \mu\alpha.c \parallel e \rangle \xrightarrow{*} \langle \mu\alpha.c' \parallel e' \rangle \longrightarrow c'[\alpha \leftarrow e']$
  then $\langle \mu\alpha.c \parallel e \rangle \longrightarrow c[\alpha \leftarrow e] \xrightarrow{*} c'[\alpha \leftarrow e']$.

- *If* $\langle r \parallel \tilde{\mu}x.c\rangle \xrightarrow{*} \langle r' \parallel \tilde{\mu}x.c'\rangle \longrightarrow c'[x \leftarrow r']$
  then $\langle r \parallel \tilde{\mu}x.c\rangle \longrightarrow c[x \leftarrow r] \xrightarrow{*} c'[x \leftarrow r']$.

3. Intersection types in $\bar{\lambda}\mu\tilde{\mu}$-calculus

The classical sequent calculus provides the framework for the definition of a type-assignment system for $\bar{\lambda}\mu\tilde{\mu}$ using simple types. This is precisely the type system of Curien and Herbelin [14], which will be the foundation upon which we build our intersection types.

Definition 2. The set $\mathbb{T}$ of *raw types* is generated from an infinite set $TVar$ of type-variables as follows.

$$\mathbb{T} ::= TVar \mid \mathbb{T} \to \mathbb{T} \mid \mathbb{T}^\circ \mid \mathbb{T} \cap \mathbb{T}$$

A° is said to be the *dual* type of type A.

We consider raw types modulo the equality generated by saying that

- intersection is associative and commutative
- for all raw types A, $A^{\circ\circ} = A$,

A *type* is either a *term-type* or a *coterm-type* or the special constant $\bot$, where the *term-types* and *coterm-types* are defined as follows.

A raw type is a term-type if it is either a type variable, or of the form $(A_1 \to A_2)$ or $(A_1 \cap \cdots \cap A_k)$, $i \geq 2$ for term-types A_i, or of the form D° for a coterm-type D. A raw type is a coterm-type if it is either a coterm variable, or of the form A° for a term-type A or of the form $(D_1 \cap \cdots \cap D_k)$, $i \geq 2$ for coterm-types D_i. Not all raw types fall into these categories: consider $\tau \cap \tau^\circ$. Note that every coterm type is a type of the form A°, where A is a term-type, or an intersection of such types.

The following taxonomy of types will be used frequently; the proof is straightforward.

Lemma 3. *Each type other than $\bot$ is uniquely – up to the equivalences mentioned above – of one of the forms in the table below. Furthermore, for each type T there is a unique type which is T°. If T is a term-type [resp., coterm-type], then T° is a coterm-type [resp., term-type].*

	term-types	*coterm-types*
	τ	τ°
	$(A_1 \to A_2)$	$(A_1 \to A_2)^\circ$
for $n \geq 2$:	$(A_1 \cap A_2 \cap \cdots \cap A_n)$	$(A_1 \cap A_2 \cap \cdots \cap A_n)^\circ$
for $n \geq 2$:	$(A_1{}^\circ \cap A_2{}^\circ \cap \cdots \cap A_n{}^\circ)^\circ$	$(A_1{}^\circ \cap A_2{}^\circ \cap \cdots \cap A_n{}^\circ)$

The characterization of the two columns as being "term-types" or "coterm-types" holds under the convention that the A_1 displayed are all term-types.

Notation. Sometimes it will be convenient to refer to types of the form $(A \to B)$ and $(A_1 \cap \cdots \cap A_k) \to B$ uniformly. For such types, we will use the notation $(\bigcap A_i \to B)$, with the understanding that the $\bigcap A_i$ might refer to a single non-intersection type.

Definition 4. A *basis* Σ is a set of statements of the form $(x : A)$ and $(\alpha : D)$ where A is a term-type, D is a coterm-type, and all (co)variables are distinct.

In the presentation of the typing rules in Fig. 1, v is any (co)variable.

Definition 5 (*Typing Rules of the System $\mathcal{M}^\cap$*). The type assignment system $\mathcal{M}^\cap$is given by the typing rules in Fig. 1.

The normal form $\lambda x.\mu\alpha.\langle x \parallel x \bullet \alpha\rangle$, which corresponds to the normal form $\lambda x.xx$ in λ calculus, is not typable in $\bar{\lambda}\mu\tilde{\mu}$ with simple types. It is typable in the currently introduced system $\mathcal{M}^\cap$ by $\lambda x.\mu\alpha.\langle x \parallel x \bullet \alpha\rangle : A \cap (A \to B) \to B$.

$$\frac{}{\Sigma,\ v : (T_1 \cap \cdots \cap T_k) \vdash v : T_i}\ (\mathrm{ax})$$

$$\frac{\Sigma,\ x : A \vdash r : B}{\Sigma \vdash \lambda x.r : A \to B}\ (\to \mathrm{r}) \qquad \frac{\Sigma \vdash r : A_i \quad i = 1,\ldots,k \qquad \Sigma \vdash e : B^\circ}{\Sigma \vdash r \bullet e : ((A_1 \cap \cdots \cap A_k) \to B)^\circ}\ (\to \mathrm{e})$$

$$\frac{\Sigma,\ \alpha : A^\circ \vdash c : \bot}{\Sigma \vdash \mu\alpha.c : A}\ (\mu) \qquad \frac{\Sigma,\ x : A \vdash c : \bot}{\Sigma \vdash \tilde{\mu}x.c : A^\circ}\ (\tilde{\mu})$$

$$\frac{\Sigma \vdash r : A \qquad \Sigma \vdash e : A^\circ}{\Sigma \vdash \langle r \parallel e \rangle : \bot}\ (\mathrm{cut})$$

Fig. 1. The typing system $\mathcal{M}^\cap$.

3.1. Discussion

In the system presented here, there is no unrestricted $\cap$-introduction rule. The significance of this will emerge during the treatments of Subject Reduction (see Section 4.2) and Type Soundness (see Section 6.4). Our types are similar in spirit to the "strict" types of van Bakel [43,44]. But things are more subtle here due to the μ and $\tilde{\mu}$ expressions: we cannot completely avoid deriving expressions which have an intersection type. Note that the pattern of the types of the derived typings in the figure matches the table in Lemma 3.

As suggested in the Introduction, the presence of intersection types in a symmetric calculus like $\overline{\lambda}\mu\tilde{\mu}$ tempts us to include union types as well. If a term has type $A \cap B$, meaning that it denotes values which inhabit both A and B, then it can interact with any continuation that can receive an A-value *or* a B-value: such a continuation will naturally be expected to have the type $A \cup B$. But any type that can be the type of a variable can be the type of a coterm (via the $\tilde{\mu}$-construction), and any type that can be the type of a covariable can be the type of a term (via the μ-construction). This suggests having intersections *and* unions for terms and continuations. But in light of the incompatibility between union types and Subject Reduction, we resist incorporating explicit union types. The use of an explicit involution operator allows us to record the relationship between an intersection $(A \cap B)$ and its dual type $(A \cap B)^\circ$. The "classical" nature of the underlying logic is reflected in the "double-negation elimination" type equality $T^{\circ\circ} = T$. But there is no identification between $(A \cap B)^\circ$ and a union of A° and B°.

4. Properties of the type system

Definition 6. If Σ_1 and Σ_2 are bases, define $\Sigma_1 \sqcap \Sigma_2$ to be

$$\begin{aligned}\Sigma_1 \sqcap \Sigma_2 = &\{v : T \mid (v : T) \in \Sigma_1 \text{ and } v \notin \Sigma_2\}\\ &\cup \{v : T \mid (v : T) \in \Sigma_2 \text{ and } v \notin \Sigma_2\}\\ &\cup \{v : T_1 \cap T_2 \mid (v : T_1) \in \Sigma_1 \text{ and } (v : T_2) \in \Sigma_2\}\end{aligned}$$

The following lemma is a straightforward induction over typing derivations.

Lemma 7 (*Basis Lemma*).

(1) *(Basis expansion) Let $\Sigma \subseteq \Sigma'$. If $\Sigma \vdash t : T$ then $\Sigma' \vdash t : T$.*
(2) *(Basis restriction) If $\Sigma \vdash t : T$ then $\Sigma \restriction_{Fv(t)} \vdash t : T$.*
(3) *(Basis intersection) Let Σ_1 and Σ_2 be bases. If $\Sigma_1 \vdash t : T$ then $\Sigma_1 \sqcap \Sigma_2 \vdash t : T$.*

A consequence of part 3 above is that if t and u are two typable expressions then, without loss of generality, we may assume that there is a single basis Σ which types each of them. Henceforth we will use this fact without explicit reference to the lemma.

The following lemma is straightforward since the typing is syntax-directed.

Lemma 8 (*Generation Lemma*).

(1) *If $\Sigma \vdash (\lambda x.r) : T$ then $T \equiv \bigcap A_i \to B$ and $\Sigma, x : \cap A_j \vdash r : B$ for some $j = 1, \ldots, n$.*
(2) *If $\Sigma \vdash (r \bullet e) : T^\circ$ then $T \equiv \bigcap A_i \to B$ and $\Sigma \vdash r : A_i$ for all i and $\Sigma \vdash e : B^\circ$.*

(3) *If* $\Sigma \vdash (\mu\alpha.c) : A$ *then* $\Sigma, \alpha : A^\circ \vdash c : \bot$.
(4) *If* $\Sigma \vdash (\widetilde{\mu}x.c) : A^\circ$ *then* $\Sigma, x : A \vdash c : \bot$.
(5) *If* $\Sigma \vdash \langle r \parallel e\rangle : \bot$, *then there exists a type* A *such that* $\Sigma \vdash r : A$ *and* $\Sigma \vdash e : A^\circ$.

4.1. Subject reduction

The type assignment system $\mathcal{M}^\cap$ enjoys the Subject Reduction property. Although this is a typical – and crucially important – property of types systems, it was difficult to achieve in a system designed to characterize strong normalization in $\overline{\lambda}\mu\widetilde{\mu}$. In the discussion following the Subject Reduction Theorem below, we point out the problem with an earlier system [19], and explain how the present system avoids it.

Lemma 9 (*Substitution*). *Let* t *and* s *be arbitrary terms or coterms, let* v *be a variable or covariable, and let* $D = (D_1 \cap \cdots \cap D_k)$. *If* $\Sigma, v : D \vdash t : T$ *and* $\Sigma \vdash s : D_i$ *for each* $i = 1, \ldots, k$ *then* $\Sigma \vdash t[v \leftarrow s] : T$.

Proof. By case on the structure of t. In what follows we assume $\Sigma \vdash s : D_i$ for all $i = 1, \ldots, k$.

t is a variable w with $w \not\equiv v$. Note that $t[v := s] \equiv w$. By assumption $\Sigma, v : D \vdash w : T$. Then by Basis restriction Lemma 7(2) $\Sigma \vdash w : T$.

t is v. Note that $t[v := s] \equiv s$, $T \equiv D_i$ for all $i = 1, \ldots, k$. Then $\Sigma, v : D \vdash v : D_i$ for all i holds by (ax). Hence the result is the second hypothesis.

$t : T$ is $\lambda x.b : \bigcap A_i \to B$. Note that by the Generation Lemma $T \equiv \bigcap A_i \to B$. Let us assume $\Sigma, v : D \vdash \lambda x.b : \bigcap A_i \to B$. Then by the Generation Lemma $\Sigma, x : \cap A_j, v : D \vdash b : B$, for some j. By the induction hypothesis $\Sigma, x : \cap A_j \vdash b[v := s] : B$ for some j. Then by the Basis intersection Lemma $\Sigma, x : \bigcap A_i \vdash b[v := s] : B$ and by ($\to$ r) we have $\Sigma \vdash \lambda x.b[v := s] : \bigcap A_i \to B$. The result follows since $t[v := s] \equiv \lambda x.b[v := s]$.

$t : T$ is $r \bullet e : (\bigcap A_i \to B)^\circ$. Let us assume $\Sigma, v : D \vdash r \bullet e : (\bigcap A_i \to B)^\circ$. Then by the Generation Lemma $\Sigma, v : D \vdash r : A_i$, for all i, and $\Sigma, v : D \vdash e : B^\circ$. By induction $\Sigma \vdash r[v := s] : A_i$, for all i, and $\Sigma \vdash e[v := s] : B^\circ$.
By ($\to$ e), $\Sigma \vdash r[v := s] \bullet e[v := s] : (\bigcap A_i \to B)^\circ$. The result follows since $t[v := s] \equiv r[v := s] \bullet e[v := s]$.

$t : T$ is $\mu\alpha.c : A$. By assumption and by the Generation Lemma we have $\Sigma, v : D, \alpha : A^\circ \vdash c : \bot$. Then by induction $\Sigma, \alpha : A^\circ \vdash c[v := s] : \bot$. The result follows by (μ) rule.

$t : T$ is $\langle r \parallel e\rangle : \bot$. We have $T \equiv \bot$ and by hypothesis $\Sigma, v : D \vdash r : A$ and $\Sigma, v : D \vdash e : A^\circ$. By induction $\Sigma \vdash r[v := s] : A$ and $\Sigma \vdash e[v := s] : A^\circ$. Hence, $\Sigma \vdash \langle r \parallel e\rangle[v := s] : \bot$. □

Theorem 10 (*Subject Reduction*). *Let* s *be any expression. If* $\Sigma \vdash s : S$ *and* $s \to s'$ *then* $\Sigma \vdash s' : S$.

Proof. The proof is by induction on s. If the redex of the reduction is not s itself then we may simply invoke the induction hypothesis. Otherwise s is a command $c = \langle r_1 \parallel e_1\rangle$ which undergoes a reduction by one of the rules (λ), (μ), or ($\widetilde{\mu}$).

By the Generation Lemma 8(5) we have, for some type T,

$$\Sigma \vdash r_1 : T \text{ and } \Sigma \vdash e_1 : T^\circ.$$

Case (λ) reduction: $c \equiv \langle \lambda x.r \parallel r' \bullet e\rangle \longrightarrow \langle r[x \leftarrow r'] \parallel e\rangle$. By the Generation Lemma 8(1) $T \equiv \bigcap A_i \to B$, $i = 1, \ldots, n$ and $\Sigma, x : \cap A_l \vdash r : B$ for some $l \in \{1, \ldots, n\}$. Then by the Basis intersection lemma $\Sigma, x : \bigcap A_i \vdash r : B$. On the other hand, by the Generation Lemma 8(2), $\Sigma \vdash r' : A_i$ for all $i = 1, \ldots, n$ and $\Sigma \vdash e : B^\circ$. By Substitution Lemma 9 $\Sigma \vdash r[x \leftarrow r'] : B$. The desired result $\Sigma \vdash \langle r[x \leftarrow r'] \parallel e\rangle : \bot$ follows by the cut rule.

Case (μ-red) reduction: $c \equiv \langle \mu\alpha.c' \parallel e\rangle \longrightarrow c'[\alpha \leftarrow e]$ and ($\widetilde{\mu}$) reduction: $c \equiv \langle r \parallel \widetilde{\mu}x.c'\rangle \longrightarrow c'[x \leftarrow r]$. Straightforward application of the Generation and the Substitution Lemma.

Case ($\widetilde{\mu}$-red) reduction: similar to the previous case. □

4.2. Discussion

It is instructive to analyze the difficulty in proving Subject Reduction for an intersection type system for $\overline{\lambda}\mu\widetilde{\mu}$ in the presence of the standard rule for intersection-introduction:

$$\frac{\Sigma \vdash t : A \qquad \Sigma \vdash t : B}{\Sigma \vdash t : (A \cap B)} (\vdash \cap)$$

specifically, its interaction with the (μ) and $(\widetilde{\mu})$ rules. Suppose that the judgment $\Sigma \vdash (\mu\alpha.c) : (A \cap B)$ is derived as follows

$$\frac{\dfrac{\Sigma,\ \alpha : A^\circ \vdash c : \bot}{\Sigma \vdash \mu\alpha.c : A}(\mu) \qquad \dfrac{\Sigma,\ \alpha : B^\circ \vdash c : \bot}{\Sigma \vdash \mu\alpha.c : B}(\mu)}{\Sigma \vdash (\mu\alpha.c) : (A \cap B)} (\vdash \cap).$$

Now suppose that we derive the judgment $\Sigma \vdash e : (A \cap B)^\circ$, and therefore derive the judgment $\Sigma \vdash \langle(\mu\alpha.c) \parallel e\rangle : \bot$. If we try to argue that the result of a μ-reduction is well-typed, by applying the Substitution Lemma, we are stuck. The proper sub-derivations of the judgment $\Sigma \vdash (\mu\alpha.c) : (A \cap B)$ do not support an argument that e can be substituted for α. (It was this case that was overlooked in [19].)

In the type system presented in this paper, the above derivation is blocked: intersection types can be generated for redexes by the (μ) or $(\widetilde{\mu})$ rules only. The rationale behind the new type system is to accept the introduction of an intersection only at specific positions and specific times when typing an expression, namely when an arrow is introduced on the left; then a type intersection is only introduced at the parameter position. Still, the new system still types exactly all the strongly normalizing expressions (Section 5).

Interestingly, the absence of the traditional $(\vdash \cap)$ rule is crucial to our treatment of the Strong Normalization Theorem in Section 6. See the remarks following Theorem 28. The same kind of restriction on the introduction of intersection in types has been used in [9,6] in work on developing type inference algorithms for intersection types.

4.3. Subject expansion

The following can be viewed as a converse of the Substitution Lemma; it is the essential ingredient in the Subject Expansion Theorem below.

Lemma 11. *Let t and s be arbitrary terms or coterms and let v be a variable or covariable. Suppose $\Sigma \vdash t[v \leftarrow s] : T$ and suppose that s is typable in context Σ. Then there is a type $D = (D_1 \cap \cdots \cap D_k)$, $k \geq 1$, such that*

$$\Sigma \vdash s : D_i \quad \textit{for each } i \quad \textit{and} \quad \Sigma, v : D \vdash t : T.$$

Proof. The proof is by induction on t. We examine the various cases for t and s.

- Suppose t is a command $\langle r \parallel e\rangle$, v is a variable x and s is a term. We have $\langle r \parallel e\rangle[x \leftarrow s] \equiv \langle r[x \leftarrow s] \parallel e[x \leftarrow s]\rangle$; hence the last typing rule applied is (cut). Therefore, for some B $\Sigma \vdash r[x \leftarrow s] : B$ and $\Sigma \vdash e[x \leftarrow s] : B^\circ$ By the induction hypothesis there exists a type D_1 such that $\Sigma \vdash s : D_1$ and $\Sigma, x : D_1 \vdash r : B$ and there exists a type D_2 such that $\Sigma \vdash s : D_2$ and $\Sigma, x : D_2 \vdash e : B^\circ$ Applying Basis intersection Lemma 7(3) twice, one gets on one side $\Sigma, x : D_1 \cap D_2 \vdash r : B$ and on the other side $\Sigma, x : D_1 \cap D_2 \vdash e : B^\circ$ and $D_1 \cap D_2$ is a type. Using (cut) we conclude $\Sigma, x : D_1 \cap D_2 \vdash \langle r \parallel e\rangle : \bot$
- When t is a command $\langle r \parallel e\rangle$, v is a covariable α and s is a coterm, the proof is similar.
- Suppose t is a term r, v is a variable x and s is a term. We examine cases for r.

  Suppose $r \equiv x$. Then $r[x \leftarrow s] \equiv s$ and we simply take D to be B, and the result follows.

  Suppose $r \equiv y \not\equiv x$. Then $r[x \leftarrow s] \equiv y$ and so $\Sigma \vdash y : B$ by assumption. Take D to be the assumed type for s under Σ and since we may expand contexts and preserve typings the result follows.

  Suppose $r \equiv \lambda y.r'$. Then $(\lambda y.r')[x \leftarrow s] \equiv \lambda y.(r'[x \leftarrow s])$, where we may assume y is not free in s.

  By assumption $\Sigma \vdash \lambda y.(r'[x \leftarrow s]) : B$ According to Generation Lemma 8 $B \equiv \bigcap T_i \to T$ and $\Sigma, y : \cap T_j \vdash r'[x \leftarrow s] : T$. for some j. By the induction hypothesis there is an $D = (D_1 \cap \cdots \cap D_s)$ such

that $\Sigma, y : \cap T_j \ \vdash \ s : D_i$ and $\Sigma, x : D, y : \cap T_j \ \vdash \ r' : T$. Then by the Basis restriction lemma $\Sigma \ \vdash \ s : D_i$ for all i, since y is not free in s, and by the Basis intersection lemma $\Sigma, x : D, y : \bigcap T_i \ \vdash \ r' : T$ Therefore $\Sigma, x : D \ \vdash \ \lambda y.r' : \bigcap T_i \to T$ as desired.

Suppose $r \equiv \mu\alpha.c$. Then $(\mu\alpha.c)[x \leftarrow s] \equiv \mu\alpha.c[x \leftarrow s]$, where α is not free in s. We have $\Sigma \ \vdash \ \mu\alpha.(c[x \leftarrow s]) : B$ The last inference in this derivation is an application of (μ). Then the argument is analogous to the previous case.

- Suppose t is a coterm e, v is a variable α and s is a coterm. Therefore $e \equiv \alpha$, $e \equiv \beta \not\equiv \alpha$ and $e \equiv r \bullet e'$.

  Suppose $t \equiv r \bullet e'$. The applied rule is $(\to \text{e})$, and there exists an Σ and A_i's such that $\Sigma \vdash r[v \leftarrow s] : A_i$, $i = 1, \ldots, m$. There exists also a B such that $\Sigma \vdash e'[v \leftarrow s] : B^\circ$. By induction, on the one hand for each $i = 1, \ldots, m$ there exists a type $D_i = (D_i^1 \cap \cdots \cap D_i^s)$ such that $\Sigma \ \vdash \ s : D_i^j$ and $\Sigma, v : D_i \ \vdash \ r : A_i$ and on the other hand there exists a type $E = (E_1 \cap \cdots \cap E_p)$ such that $\Sigma \ \vdash \ s : E_l$ for all $l = 1, \ldots p$ and $\Sigma, v : E \ \vdash \ e' : B$. Setting $D = (D_1 \cap \cdots \cap D_m \cap E)$, we conclude the proof by Lemma 7(3). □

5. Strongly normalizing expressions are typable

Theorem 12. *Let t be an expression in normal form. Then there is a basis Σ and a type T such that $\Sigma \ \vdash \ t : T$.*

Proof. The proof is by induction on expressions.

Note that a normal form is one of the following:

$$x \qquad \alpha \qquad \lambda x.r \qquad r \bullet e \qquad \mu\beta.c \qquad \widetilde{\mu} y.c \qquad \langle r \parallel \alpha \rangle \qquad \langle x \parallel e \rangle$$

where r and e are normal forms and in the latter two cases r is not $\mu\beta.c$ and e is not $\widetilde{\mu} y.c$.

Of course variables x and α are immediately typable. If t is $\lambda x.r$, then by induction we have $\Sigma \ \vdash \ r : B$ Without loss of generality we may assume a binding $x : D$ in Σ. Then we have $\Sigma \setminus x : D \ \vdash \ \lambda x.r : D \to B$

If t is $r \bullet e$, we have by induction $\Sigma_1 \ \vdash \ r : A$ and $\Sigma_2 \ \vdash \ e : B^\circ$ By the Basis intersection lemma

$$\Sigma_1 \sqcap \Sigma_2 \ \vdash \ r : A \quad \text{and} \quad \Sigma_1 \sqcap \Sigma_2 \ \vdash \ e : B^\circ$$

Then $\Sigma_1 \sqcap \Sigma_2 \ \vdash \ r \bullet e : (A \to B)^\circ$

If t is $\mu\alpha.c$, then by induction we have $\Sigma \ \vdash \ c : \bot$ Without loss of generality, we may assume a binding $\alpha : D$ in Σ. Then $\Sigma \setminus \alpha : D \ \vdash \ (\mu\alpha.c) : D^\circ$.

If t is $\widetilde{\mu} x.c$ the argument is similar.

If t is $\langle r \parallel \alpha \rangle$ then by induction we have, for some Σ and D, $\Sigma \ \vdash \ r : D$ Let Σ^* be $\Sigma \sqcap \{(\alpha : D^\circ)\}$. Then $\Sigma^* \ \vdash \ r : D$ and $\Sigma^* \ \vdash \ \alpha : D^\circ$ So $\Sigma^* \ \vdash \ \langle r \parallel \alpha \rangle : \bot$

The case of $\langle x \parallel e \rangle$ is similar. □

Theorem 13 (*SN Implies Typability*). *If t is strongly normalizing, then t is typable.*

Proof. The proof is by induction over the length of the longest reduction sequence out of t, with a sub-induction on the size of t.

If t is a normal form, then t is typable by Theorem 12.

Next, suppose that t is an expression which is not itself a redex. Given that t is not a normal form, it is of one of the following forms

$$\lambda x.r \qquad r \bullet e \qquad \langle \lambda x.r \parallel \alpha \rangle \qquad \langle x \parallel r \bullet e \rangle$$

By the induction hypothesis, each r and e above is typable; it is straightforward to build a typing for t in each case.

Finally, suppose that t is a command c which is itself a redex.

If t is of the form $\langle \mu\alpha.c \parallel e \rangle$ or $\langle r \parallel \widetilde{\mu} x.r \rangle$, then let t' be obtained by doing a top-level reduction. Then either $t' = c[\alpha \leftarrow e]$ or $t' = c[x \leftarrow r]$, and by induction t' is typable under some context. Note that the expressions e and r are each strongly normalizing and e has a lower induction measure than $\langle \mu\alpha.c \parallel e \rangle$; also r has a lower induction measure than $\langle r \parallel \widetilde{\mu} x.c \rangle$, so by induction e, respectively r, is typable under some context. Then an application of Lemma 11 yields a typing for t.

If t is of the form $\langle \lambda x.r \parallel s \bullet e \rangle$, then let t' be $\langle r[x \leftarrow s] \parallel e \rangle$. Note that t reduces to t' (in two steps).

By the induction hypothesis, there is a Σ with $\Sigma \vdash \langle r[x \leftarrow s] \parallel e\rangle : \bot$. In this typing we may assume, without loss of generality, that (cut) has been the last rule applied. Therefore for some type B $\Sigma \vdash r[x \leftarrow s] : B$ and $\Sigma \vdash e : B^\circ$. Note also that by induction s is typable. Without loss of generality, we may suppose it is typable under the same context Σ. Then by Lemma 11, there is a type $D = (D_1 \cap \cdots \cap D_n)$ such that

$$\Sigma \vdash s : D_i \quad \text{and} \quad \Sigma, x : D \vdash r : B$$

From the typings $\Sigma, x : D \vdash r : B$ and $\Sigma \vdash s : D_i$ and $\Sigma \vdash e : B^\circ$, it is easy to construct a typing for $\langle \lambda x.r \parallel s \bullet e\rangle$. □

6. Typable expressions are SN

Here we give a proof of strong normalization under free reduction, for typable expressions. The difficulty in proving SN in $\overline{\lambda}\mu\widetilde{\mu}$ using a traditional reducibility (or "candidates") argument arises from the critical pairs $\langle \mu\alpha.c \parallel \widetilde{\mu}x.d\rangle$. Since neither of the expressions here can be identified as the preferred redex, one cannot define candidates by induction on the structure of types. This difficulty arises already in the simply-(arrow)-typed case. The "symmetric candidates" technique in [1,35] uses a fixed-point technique to define the candidates and suffices to prove strong normalization for simply-typed $\overline{\lambda}\mu\widetilde{\mu}$.

The interaction between intersection types and symmetric candidates is technically problematic (see [17] for a discussion about a related calculus). The discussion just after the proof of Theorem 28 explains how this problem is addressed in the current type system.

6.1. Pairs

Let Λ_r denote the set of all terms and Λ_e denote the set of all coterms.

Definition 14. A *pair* is given by two sets R and E with $R \subseteq \Lambda_r$ and $E \subseteq \Lambda_e$, each of which is non-empty. The pair (R, E) is *stable* if for every $r \in R$ and every $e \in E$, the command $\langle r \parallel e\rangle$ is SN.

For example, the pair (Var_r, Var_e) is stable. Since the sets in a pair are non-empty, any stable pair consists of SN expressions.

The following technical condition will be crucial to the use of pairs to interpret types (it is this condition which makes the Type Soundness Theorem go through, specifically the cases of typing a μ or a $\widetilde{\mu}$ expressions).

Definition 15. A pair (R, E) is *saturated* if

- whenever $\mu\alpha.c$ satisfies: $\forall e \in E, c[\alpha \leftarrow e]$ is SN, then $\mu\alpha.c \in R$, and
- whenever $\widetilde{\mu}x.c$ satisfies: $\forall r \in R, c[x \leftarrow r]$ is SN, then $\widetilde{\mu}x.c \in E$.

We can always expand a pair to be saturated. It is more delicate to expand a stable pair to be saturated and remain stable. The development below achieves this. The technique is similar to the "symmetric candidates" technique as used by Barbanera and Berardi [1] for the Symmetric Lambda Calculus and adapted by Polonovski [35] in his proof of strong normalization for $\overline{\lambda}\mu\widetilde{\mu}$ calculus with explicit substitutions.

Definition 16. An expression is *simple* if it is not of the form $\mu\alpha.c$ or $\widetilde{\mu}x.c$. A set $R \subseteq \Lambda_r$ is simple if each term in R is simple; a set $E \subseteq \Lambda_e$ is simple if each coterm in E is simple.

Definition 17. Define the maps $\Phi_r : 2^{\Lambda_e} \to 2^{\Lambda_r}$ and $\Phi_e : 2^{\Lambda_r} \to 2^{\Lambda_e}$ by

$$\begin{aligned}\Phi_r(Y) &= \{r \mid r \text{ is of the form } \mu\alpha.c \text{ and } \forall e \in Y, c[\alpha \leftarrow e] \text{ is SN}\}\\ &\quad \cup \{r \mid r \text{ is simple and } \forall e \in Y, \langle r \parallel e\rangle \text{ is SN}\}\\ \Phi_e(X) &= \{e \mid e \text{ is of the form } \widetilde{\mu}x.c \text{ and } \forall r \in X, c[x \leftarrow r] \text{ is SN}\}\\ &\quad \cup \{e \mid e \text{ is simple and } \forall r \in X, \langle r \parallel e\rangle \text{ is SN}\}\end{aligned}$$

Note that if $Y \neq \emptyset$, then $\Phi_r(Y) \subseteq SN$, and if $X \neq \emptyset$ then $\Phi_e(X) \subseteq SN$. Also, if $Y \subseteq SN$ then all term variables are in $\Phi_r(Y)$, and if $X \subseteq SN$, then all coterm variables are in $\Phi_e(X)$.

It is easy to see that each of Φ_e and Φ_r is antimonotone. So the maps $(\Phi_r \circ \Phi_e) : \Lambda_r \to \Lambda_r$ and $(\Phi_e \circ \Phi_r) : \Lambda_e \to \Lambda_e$ are monotone. By the Knaster–Tarski fixed point theorem [29,41], each of these maps has a complete lattice of fixed points, ordered by set inclusion.

The following lemma is straightforward.

Lemma 18. *Let (R, E) be a pair. The following are equivalent.*

- *$\Phi_e(R) = E$ and $\Phi_r(E) = R$.*
- *R is a fixed point for $(\Phi_r \circ \Phi_e)$ and $E = \Phi_e(R)$.*
- *E is a fixed point for $(\Phi_e \circ \Phi_r)$ and $R = \Phi_r(E)$.*

Definition 19. A pair (R, E) is a *mutual fixed point* for Φ_r and Φ_e if any of the conditions of Lemma 18 hold.

Lemma 20. *Suppose (R, E) is a mutual fixed point for Φ_r and Φ_e. Then (R, E) is stable and saturated.*

Proof. Each of R and E is non-empty since, as images of Φ_r and Φ_e, they contain all variables. It follows that each expression in R or E is SN.

Saturation follows immediately from the facts that $\Phi_e(R) = E$ and $\Phi_r(E) = R$.

For stability: consider any command $\langle r \parallel e\rangle$ with $r \in R$ and $e \in E$; we must show that this command is SN. Since R and E is a set of SN expression, it suffices, by Lemma 1, to show that the result of a top-level reduction is SN. But such a reduction step can only be one of $\langle r \parallel e\rangle \equiv \langle \mu\alpha.c \parallel e\rangle \longrightarrow c[\alpha \leftarrow e]$ or $\langle r \parallel e\rangle \equiv \langle r \parallel \widetilde{\mu}x.c\rangle \longrightarrow c[x \leftarrow r]$. In the first case we have SN because of the fact that $r \in R = \Phi_r(E)$. In the second case we have SN because of the fact that $e \in E = \Phi_e(R)$. □

So now our task is to show how to make mutual fixed points which have the right structure for interpreting types. The strategy for defining saturated pairs for types is slightly different depending on whether the type to be interpreted is: (i) an arrow-type or its dual; or (ii) an intersection or its dual. In the former case, we need to establish that the operators $(\Phi_r \circ \Phi_e)$ and $(\Phi_e \circ \Phi_r)$ are inflationary on strongly normalizing simple sets.

Lemma 21. *If R is a simple set of SN terms then $R \subseteq \Phi_r(\Phi_e(R))$, and similarly for simple coterms.*

Proof. We treat only the assertion about terms. Let $r \in R$; it suffices to show that for all $e \in \Phi_e(R)$, $\langle r \parallel e\rangle$ is SN. Let $e \in \Phi_e(R)$: if e is simple then $\langle r \parallel e\rangle$ is SN by definition of $\Phi_e(R)$. Otherwise e is $\widetilde{\mu}x.d$ and we have $d[x \leftarrow r]$ is SN. We wish to show that $\langle r \parallel \widetilde{\mu}x.d\rangle$ is SN. Since r and d are SN, it suffices by Lemma 1 to show the result of a top-level reduction is SN; but since r is simple this must be $d[x \leftarrow r]$. □

As is well-known, when $\mathcal{G}$ is a monotone operator on a complete lattice of sets and X satisfies $X \subseteq \mathcal{G}(X)$, then

$$fix_X^{\mathcal{G}} = \bigcap\{Y \mid X \subseteq Y \wedge \mathcal{G}(Y) \subseteq Y\}$$

is a fixed point of $\mathcal{G}$ with $X \subseteq fix_X^{\mathcal{G}}$. This, in light of Lemma 21, justifies the following definition.

Definition 22. If R is a simple set of SN terms, let $R^{\uparrow}$ be the least fixed point of $(\Phi_r \circ \Phi_e)$ with the property that $R \subseteq R^{\uparrow}$.

If E is a simple set of SN coterms, let $E^{\uparrow}$ be the least fixed point of $(\Phi_e \circ \Phi_r)$ with the property that $E \subseteq E^{\uparrow}$.

The following is an immediate consequence of Lemma 18.

Lemma 23. *If R is a simple set of SN terms then $(R^{\uparrow}, \Phi_e(R^{\uparrow}))$ is a mutual fixed point of Φ_r and Φ_e, with $R \subseteq R^{\uparrow}$.*
Similarly, if E is a simple set of SN coterms, then $(\Phi_r(E^{\uparrow}), E^{\uparrow})$ is a mutual fixed point of Φ_r and Φ_e, with $E \subseteq E^{\uparrow}$.

In Definition 25, we will use the above construction to interpret types which are not intersections (or their duals). When the types we want to interpret are intersections, or types of the form $(T_1 \cap \cdots \cap T_k)^{\circ}$, the above construction does not work. The essential problem is that the intersection of saturated pairs does not in general yield a saturated pair. This means that the interpretation of an intersection type $(A \cap B)$ will not be the intersection of the interpretations of A and B. But the collection of fixed points of $(\Phi_r \circ \Phi_e)$ (and that of $(\Phi_e \circ \Phi_r)$) carries its own lattice structure under inclusion, and this is all we require to interpret intersection types.

Definition 24. Let $\mathsf{Fix}_{(\Phi_r \circ \Phi_e)}$ be the set of fixed points of the operator $(\Phi_r \circ \Phi_e)$. If $R_1, \ldots, R_k$ are fixed points of $(\Phi_r \circ \Phi_e)$, let $(R_1 \curlywedge \cdots \curlywedge R_k)$ denote the meet of these elements in the lattice $\mathsf{Fix}_{(\Phi_r \circ \Phi_e)}$.

Let $\mathsf{Fix}_{(\Phi_e \circ \Phi_r)}$ be the set of fixed points of the operator $(\Phi_e \circ \Phi_r)$. Let $(E_1 \curlywedge \cdots \curlywedge E_k)$ denote the meet of fixed points of $(\Phi_e \circ \Phi_r)$.

The set of objects of the lattice $\mathsf{Fix}_{(\Phi_r \circ \Phi_e)}$ is a subset of the set 2^{Λ_r}. Since each of these lattices is ordered by set inclusion, we have $(R_1 \curlywedge \cdots \curlywedge R_k) \subseteq R_i$ for each i. Since $\cap$ is the greatest lower bound operator in 2^{Λ_r},

$$(R_1 \curlywedge \cdots \curlywedge R_k) \subseteq (R_1 \cap \cdots \cap R_k).$$

Similarly for the meet in $\mathsf{Fix}_{(\Phi_e \circ \Phi_r)}$.

We stress that $(R_1 \curlywedge \cdots \curlywedge R_k)$ is a fixed point of $(\Phi_r \circ \Phi_e)$ and so the pair

$$((R_1 \curlywedge \cdots \curlywedge R_k), \quad \Phi_e(R_1 \curlywedge \cdots \curlywedge R_k))$$

is a mutual fixed point of Φ_r and Φ_e.

6.2. Pairs and types

For each type T, we define the set $[\![T]\!]$; when T is a term (respectively, coterm) type, then $[\![T]\!]$ will be a set of terms (respectively, coterms).

Guided by Lemma 3, we will define $[\![T]\!]$ and $[\![T^\circ]\!]$ simultaneously.

Definition 25 (*Interpretation of Types*). For each type T we define the set $[\![T]\!]$, maintaining the invariant that when T is a term type, then $[\![T]\!]$ is a fixed point of $(\Phi_r \circ \Phi_e)$, and when T is a coterm-type then $[\![T]\!]$ is a fixed point of $(\Phi_e \circ \Phi_r)$.

- When T is $\bot$, then $[\![T]\!]$ is the set of SN commands.
- When T is a type variable, we set R to be the set of term variables, and then construct the pair $(R^\uparrow, \Phi_e(R^\uparrow))$. We then take $[\![T]\!]$ to be $R^\uparrow$ and $[\![T^\circ]\!]$ to be $\Phi_e(R^\uparrow)$.
- Suppose T is $(\bigcap A_i \to B)$. Set E to be $\{r \bullet e \mid \forall i, r \in [\![A_i]\!]$ and $e \in [\![B^\circ]\!]\}$ then construct the pair $(\Phi_r(E^\uparrow), E^\uparrow)$. We then take $[\![T]\!]$ to be $\Phi_r(E^\uparrow)$ and $[\![T^\circ]\!]$ to be $(E^\uparrow)$.
- When T is $(A_1 \cap A_2 \cdots \cap A_n)$, $n \geq 2$, we take $[\![T]\!]$ to be $([\![A_1]\!] \curlywedge \cdots \curlywedge [\![A_n]\!])$, and then take $[\![T^\circ]\!]$ to be $\Phi_e([\![T]\!])$.
- When T is $(A_1{}^\circ \cap A_2{}^\circ \cdots \cap A_n{}^\circ)^\circ$, $n \geq 2$, we take $[\![T^\circ]\!]$ to be $([\![A_1{}^\circ]\!] \curlywedge \cdots \curlywedge [\![A_n{}^\circ]\!])$ and then take $[\![T]\!]$ to be $\Phi_r([\![T^\circ]\!])$.

Note that by definition, for each type T the pair $([\![T]\!], [\![T^\circ]\!])$ is a mutual fixed point of Φ_r and Φ_e and so constitutes a stable saturated pair.

The following collects the information we need to prove Type Soundness.

Lemma 26. (1) *For each type T, $[\![T]\!]$ is a set of SN (co)terms.*
(2) $[\![(\bigcap A_i \to B)^\circ]\!] \supseteq \{r \bullet e \mid \forall i, r \in [\![A_i]\!]$ *and* $e \in [\![B^\circ]\!]\}$
(3) $(\lambda x.b) \in [\![(\bigcap A_i \to B)]\!]$ *if for all r such that $\forall i, r \in [\![A_i]\!]$, we have $b[x \leftarrow r] \in [\![B]\!]$*
(4) $(\mu\alpha.c) \in [\![A]\!]$ *if for all $e \in [\![A^\circ]\!]$ we have $c[\alpha \leftarrow e]$ SN. Similarly, $(\widetilde{\mu} x.c) \in [\![A^\circ]\!]$ if for all $r \in [\![A]\!]$, we have $c[x \leftarrow r]$ SN.*
(5) $[\![(T_1 \cap \cdots \cap T_k)]\!] \subseteq ([\![T_i]\!] \cap \cdots \cap [\![T_k]\!])$

Proof. (1) The pair $([\![T]\!], [\![T^\circ]\!])$ is stable.
(2) Letting E denote to the right-hand side of the inclusion, we note that $[\![(\bigcap A_i \to B)^\circ]\!]$ is precisely $E^\uparrow$. Lemma 23 yields the result.
(3) Let $E = \{r \bullet e \mid \forall i, r \in [\![A_i]\!]$ and $e \in [\![B^\circ]\!]\}$. We first claim that $(\lambda x.b) \in \Phi_r(E)$. By Lemma 21, it suffices to show that for all $(r \bullet e) \in E$ $\langle(\lambda x.b \parallel r \bullet e\rangle$ is SN. By the previous part, each such $(r \bullet e)$ is in $[\![(\bigcap A_i \to B)^\circ]\!]$. The result then follows from the fact that the pair $([\![(\bigcap A_i \to B)]\!], [\![(\bigcap A_i \to B)^\circ]\!])$ is stable.
(4) By the fact that the pair $([\![A]\!], [\![A^\circ]\!])$ is saturated.
(5) This holds simply because $[\![(T_1 \cap \cdots \cap T_k)]\!] = ([\![T_1]\!] \curlywedge \cdots \curlywedge [\![T_k]\!])$ (cf. the observation following Definition 24). □

6.3. Soundness and strong normalization

Since each $[\![T]\!]$ consists of SN expressions, the following theorem will imply that all typable expressions are SN.

Theorem 27 (*Type Soundness*). *If expression t is typable with type T, then t is in $[\![T]\!]$.*

Proof. Let us say that a substitution θ *satisfies* Σ if the following holds for each statement $(v : T)$ of Σ: *if T is of the form $(T_1 \cap \cdots \cap T_k)$, $k \geq 1$, then for each i, $\theta v \in [\![T_i]\!]$.*

Then to prove the theorem, it is convenient to prove the following stronger statement:

If $\Sigma \vdash t : T$ and θ satisfies Σ, then $\theta t \in [\![T]\!]$.

This implies the theorem, since the identity substitution satisfies every Σ (since each $[\![T]\!]$ contains all variables and covariables). We prove the statement above by induction on typing derivations.

Choose a typing $\Sigma \vdash t : T$ and a substitution θ, that satisfies Σ; we wish to show that $\theta t \in [\![T]\!]$. We consider the possible forms of the given typing.

Case: (ax) of $\mathcal{M}^\cap$ Immediate from the fact that θ satisfies Σ, $v : (T_1 \cap \cdots \cap T_k)$.

Case: ($\to$ e) of $\mathcal{M}^\cap$ We wish to show that $\theta(r \bullet e) = (\theta r \bullet \theta e) \in [\![(\bigcap A_i \to B)^\circ]\!]$. By Lemma 26 part 2, it suffices to show that $\theta r \in [\![A_i]\!]$ for each i and $\theta e \in [\![B]\!]$, and these hold by the induction hypothesis.

Case: ($\to$ r) of $\mathcal{M}^\cap$ Allowing for the fact that A might be an intersection, write $(A \to B)$ as $(\bigcap A_i \to B)$. We wish to show that $\theta(\lambda x.b) = (\lambda x.\theta b) \in [\![\bigcap A_i \to B]\!]$. By Lemma 26 part 3, it suffices to consider an r such that for every i, $r \in [\![A_i]\!]$, and show that $\theta b[x \leftarrow a]$ is SN. Let θ' be the substitution which adds the binding $x \mapsto r$ to θ; it suffices to show that $\theta'(b) \in [\![B]\!]$. By Lemma 26 part 5 the substitution θ' satisfies Σ. So $\theta'(b) \in [\![B]\!]$ by induction.

Case: (μ) of $\mathcal{M}^\cap$ We wish to show that $(\mu\alpha.c) \in [\![A]\!]$.

By Lemma 26 part 4, it suffices to consider an arbitrary $e \in [\![A^\circ]\!]$ and show that $\theta c[\alpha \leftarrow e]$ is SN. Let θ' be the substitution which adds the binding $\alpha \mapsto e$ to θ; it suffices to show that $\theta'(c)$ is SN. The substitution θ' satisfies Σ, invoking Lemma 26 part 5 in case A° is an intersection. So $\theta'(c)$ is SN by induction. **Case: (μ) of $\mathcal{M}^\cap$** Similar to the case of (μ).

Case: (cut) of $\mathcal{M}^\cap$ We need to show that $\langle \theta r \parallel \theta e \rangle$ is SN. By induction $\theta r \in [\![B]\!]$ and $\theta e \in [\![B]\!]$, so the result follows from the stability of $[\![B]\!]$.

This completes the proof. □

Now we can prove the converse of Theorem 13.

Theorem 28 (*Typability Implies SN*). *Every typable expression is SN.*

Proof. By Theorem 27 and the fact that every $[\![T]\!]$ consists of SN expressions. □

By Theorems 13 and 28 we obtain the main result that the introduced typing system $\mathcal{M}^\cap$ completely characterizes all strongly normalizing $\overline{\lambda}\mu\widetilde{\mu}$ terms.

Corollary 29. *A $\overline{\lambda}\mu\widetilde{\mu}$ term is strongly normalizing if and only if it is typable in $\overline{\lambda}\mu\widetilde{\mu}$.*

6.4. Discussion

From a semantical perspective, the $(\vdash \cap)$ rule (see Section 4.2) asserts that the interpretation of an intersection $(A \cap B)$ must *contain* the interpretations of A and of B. In contrast, the form of the (ax) rule in our system (equivalent to an intersection-elimination rule) asserts that the interpretation of an intersection $(A \cap B)$ must *be contained in* the interpretations of A and of B. We have seen that the absence of the $(\vdash \cap)$ rule was the key to establishing the Subject Reduction result. It is very interesting to note that relaxing the semantic requirement embodied in the $(\vdash \cap)$ rule is crucial in the Soundness Theorem above as well. If one wants to interpret the type $(A \cap B)$ by a saturated set built from the interpretations $[\![A]\!]$ and $[\![B]\!]$ of A and B, one must confront the fact that the intersection of two saturated sets will not itself be saturated. It is illuminating to explore the problem of trying to build a suitable saturated set from $[\![A]\!]$ and $[\![B]\!]$. The crucial point is that one can build such a set containing the *simple* expressions in $[\![A]\!] \cap [\![B]\!]$ and altogether *contained in* $[\![A]\!] \cap [\![B]\!]$. Since our type system has abandoned the $(\vdash \cap)$ rule, this is sufficient.

7. Open problems and conclusion

We defined intersection for the $\overline{\lambda}\mu\widetilde{\mu}$-calculus extending the Dezani–Coppo heritage from the λ-calculus to the $\overline{\lambda}\mu\widetilde{\mu}$-calculus. Our system, $\mathcal{M}^{\cap}$ has the properties that the typable expressions are precisely the strongly normalizing expressions under free (unrestricted) reduction in untyped $\overline{\lambda}\mu\widetilde{\mu}$-calculus. The system enjoys type preservation under reduction (subject reduction).

There are several open problems relating to: the logical meaning of intersection in $\overline{\lambda}\mu\widetilde{\mu}$ characterization of termination properties of $\overline{\lambda}\mu\widetilde{\mu}$-expressions, extending the type system with union types, to mention just a few.

Intersection types as logical connectives. It is well known that the traditional λ-calculus with intersection types does not fit into the Curry–Howard (proofs-as-terms) correspondence. This makes the intersection a proof-theoretical and not a truth-functional connective [28]. There have been several attempts to develop a typed system (à la Church) with intersection types by Dezani et al. [18], Ronchi Della Rocca and Roversi [39], Capitani et al. [8] and recently by Wells and Haack [48]. This direction of research in the framework of $\lambda\mu$-calculus merits attention.

Termination properties. Intersection types have proven to be an invaluable tool for studying reduction properties in the traditional λ-calculus, and in future work we expect to use suitable variants on the system presented here to characterize weak normalization and head-normalization in $\overline{\lambda}\mu\widetilde{\mu}$.

Union types. Buneman and Pierce [7], have shown how union types can play a key role in the design of query languages for semistructured data union types. Union and intersection types have recently been used by Palsberg and Pavlopoulou [32] and subsequently by Wells, Dimock, Muller, and Turbak [47] in a type system involving *flow types* for encoding control and data flow information in typed program representations. The system in [47] obeys the Subject Reduction for a certain call-by-value version of the β-rule (in which variables are not considered values). In two papers [22,23], Dunfield and Pfenning investigate a type system incorporating – among others – union types. Their language is specifically a call-by-value language, and their type system and type assignment algorithms exploit this aspect in interesting ways. Union types in the framework of computational interpretations of classical logic will be another direction of our future research. Since this framework is especially noticeable for its account of call-by-value and call-by-name, an interesting research perspective could be to study how union and intersection types connected with a type directed reduction can actually implement dynamic strategies of functional program evaluation.

Acknowledgment

We are grateful for a particularly careful reading of a first version of this paper by an anonymous referee, which led to a simplification of our type system and a resulting strengthening of our results.

References

[1] F. Barbanera, S. Berardi, A symmetric lambda calculus for classical program extraction, Information and Computation 125 (2) (1996) 103–117.
[2] F. Barbanera, M. Dezani-Ciancaglini, U. de' Liguoro, Intersection and union types: Syntax and semantics, Information and Computation 119 (2) (1995) 202–230.
[3] H.P. Barendregt, The Lambda Calculus: Its Syntax and Semantics, revised edition, North-Holland, Amsterdam, 1984.
[4] H.P. Barendregt, M. Coppo, M. Dezani-Ciancaglini, A filter lambda model and the completeness of type assignment, Journal of Symbolic Logic 48 (4) (1984) 931–940. 1983.
[5] H.P. Barendregt, S. Ghilezan, Lambda terms for natural deduction, sequent calculus and cut-elimination, Journal of Functional Programming 10 (1) (2000) 121–134.
[6] G. Boudol, P. Zimmer, On type inference in the intersection type discipline, Electronic Notes in Theoretical Computer Science 136 (2005) 23–42.
[7] P. Buneman, B. Pierce, Union types for semistructured data, in: Research Issues in Structured and Semistructured Database Programming, 7th International Workshop on Database Programming Languages, in: Lecture Notes in Computer Science, vol. 1949, 2000, pp. 184–207.
[8] B. Capitani, M. Loreti, B. Venneri, Hyperformulae, parallel deductions and intersection types, in: BOTH '01, in: Electronic Notes in Theoretical Computer Science, vol. 50, Elsevier, 2001.
[9] S. Carlier, J.B. Wells, Type inference with expansion variables and intersection types in system E and an exact correspondence with beta-reduction, in: Eugenio Moggi, David Scott Warren (Eds.), PPDP, ACM Press, 2004, pp. 132–143.
[10] M. Coppo, M. Dezani-Ciancaglini, A new type-assignment for lambda terms, Archiv für Mathematische Logik 19 (1978) 139–156.
[11] M. Coppo, M. Dezani-Ciancaglini, An extension of the basic functionality theory for the λ-calculus, Notre Dame Journal of Formal Logic 21 (4) (1980) 685–693.

[12] M. Coppo, M. Dezani-Ciancaglini, B. Venneri, Principal type schemes and λ-calculus semantics, in: J.P. Seldin, J.R. Hindley (Eds.), To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, Academic Press, London, 1980, pp. 535–560.
[13] P.-L. Curien, Symmetry and interactivity in programming, Bulletin of Symbolic Logic 9 (2) (2003) 169–180.
[14] P.-L. Curien, H. Herbelin, The duality of computation, in: Proc. of the 5th ACM SIGPLAN Int. Conference on Functional Programming, ICFP'00, Montreal, Canada, ACM Press, 2000.
[15] R. David, Normalization without reducibility, Annals of Pure and Applied Logic 107 (2001) 121–130.
[16] R. David, K. Nour, Arithmetical proofs of strong normalization results for the symmetric $\lambda\mu$-calculus, in: Pawel Urzyczyn (Ed.), TLCA, in: Lecture Notes in Computer Science, vol. 3461, 2005, pp. 162–178.
[17] R. David, K. Nour, Why the usual candidates of reducibility do not work for the symmetric $\lambda\mu$-calculus, Electronic Notes in Theoretical Computer Science 140 (2005) 101–111.
[18] M. Dezani-Ciancaglini, S. Ghilezan, B. Venneri, The "relevance" of intersection and union types, Notre Dame Journal of Formal Logic 38 (2) (1997) 246–269.
[19] D. Dougherty, S. Ghilezan, P. Lescanne, Characterizing strong normalization in a language with control operators, in: Sixth ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, PPDP'04, ACM Press, 2004, pp. 155–166.
[20] D. Dougherty, S. Ghilezan, P. Lescanne, Intersection and union types in the $\bar{\lambda}\mu\tilde{\mu}$-calculus, in: M. Coppo, F. Damiani (Eds.), Intersection Types and Related Systems 2004, in: Electronic Notes in Theoretical Computer Science, vol. 136, 2005, pp. 153–172.
[21] D. Dougherty, S. Ghilezan, P. Lescanne, S. Likavec, Strong normalization of the dual classical sequent calculus, in: G. Sutcliffe, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning, 12th International Conference, LPAR 2005, in: Lecture Notes in Computer Science, vol. 3835, 2005, pp. 169–183.
[22] J. Dunfield, F. Pfenning, Type assignment for intersections and unions in call-by-value languages, in: A.D. Gordon (Ed.), Proceedings of the 6th International Conference on Foundations of Software Science and Computation Structures, FOSSACS'03, in: Lecture Notes in Computer Science, vol. 2620, April 2003, pp. 250–266.
[23] J. Dunfield, F. Pfenning, Tridirectional typechecking, in: X. Leroy (Ed.), Conference Record of the 31st Annual ACM Symp. on Principles of Programming Languages, POPL'04, Venice, Italy, ACM Press, January 2004, pp. 281–292.
[24] T. Griffin, A formulae-as-types notion of control, in: Proc. of the 19th Annual ACM Symp. on Principles Of Programming Languages, POPL'90, San Fransisco (CA, USA), ACM Press, 1990, pp. 47–58.
[25] H. Herbelin, Personal communication.
[26] H. Herbelin, Séquents qu'on calcule : de l'interprétation du calcul des séquents comme calcul de λ-termes et comme calcul de stratégies gagnantes. Thèse d'université, Université Paris 7, Janvier 1995.
[27] H. Herbelin, C'est maintenant qu'on calcule, au cœur de la dualité. Mémoire d'habiliation, Université de Paris-Orsay, December 2005.
[28] J.R. Hindley, Coppo–Dezani types do not correspond to propositional logic, Theoretical Computer Science 28 (1–2) (1984) 235–236.
[29] B. Knaster, Un théorème sur les fonctions d'ensembles, Annales de la Societé Polonaise de Mathematique 6 (1928) 133–134.
[30] S. Lengrand, Call-by-value, call-by-name, and strong normalization for the classical sequent calculus, in: Bernhard Gramlich, Salvador Lucas (Eds.), in: Electronic Notes in Theoretical Computer Science, vol. 86, 2003.
[31] S. Likavec, Types for object oriented and functional programming languages, Ph.D. Thesis, Università di Torino, Italy and ENS Lyon, France, 2005.
[32] J. Palsberg, C. Pavlopoulou, From polyvariant flow information to intersection and union types, Journal of Functional Programming 11 (3) (2001) 263–317.
[33] M. Parigot, An algorithmic interpretation of classical natural deduction, in: Proceedings of International Conference on Logic Programming and Automated Reasoning, LPAR'92, in: Lecture Notes in Computer Science, vol. 624, 1992, pp. 190–201.
[34] B.C. Pierce, Programming with intersection types, union types, and polymorphism, Technical Report CMU-CS-91-106, Carnegie Mellon University, February 1991.
[35] E. Polonovski, Strong normalization of $\lambda\mu\tilde{\mu}$-calculus with explicit substitutions, in: Igor Walukiewicz (Ed.), Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004, in: Lecture Notes in Computer Science, vol. 2987, 2004, pp. 423–437.
[36] G. Pottinger, Normalization as homomorphic image of cut-elimination, Annals of Mathematical Logic 12 (1977) 323–357.
[37] G. Pottinger, A type assignment for the strongly normalizable λ-terms, in: J.P. Seldin, J.R. Hindley (Eds.), To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, Academic Press, London, 1980, pp. 561–577.
[38] J.C. Reynolds, Design of the programming language Forsythe, Report CMU–CS–96–146, Carnegie Mellon University, Pittsburgh, PA, June 28, 1996.
[39] S. Ronchi Della Rocca, L. Roversi, Intersection logic, in: Computer Science Logic, CSL'01, in: Lecture Notes in Computer Science, vol. 2142, 2001, pp. 421–428.
[40] P. Sallé, Une extension de la théorie des types en lambda-calcul, in: G. Ausiello, C. Böhm (Eds.), Fifth International Conference on Automata, Languages and Programming, in: Lecture Notes in Computer Science, vol. 62, 1978, pp. 398–410.
[41] A. Tarski, A lattice-theoretical fixpoint theorem and its applications, Pacific Journal of Mathematics 5 (1955) 285–309.
[42] C. Urban, G.M. Bierman, Strong normalisation of cut-elimination in classical logic, in: Typed Lambda Calculus and Applications, in: Lecture Notes in Computer Science, vol. 1581, 1999, pp. 365–380.
[43] S. van Bakel, Principal type schemes for the Strict Type Assignment System, Logic and Computation 3 (6) (1993) 643–670.
[44] S. van Bakel, Intersection type assignment systems, Theoretical Computer Science 38 (2) (1997) 246–269.
[45] Ph. Wadler, Call-by-value is dual to call-by-name, in: Proc. of the 8th Int. Conference on Functional Programming, 2003, pp. 189–201.
[46] Ph. Wadler, Call-by-value is dual to call-by-name, reloaded, in: Rewriting Technics and Application, RTA'05, in: Lecture Notes in Computer Science, vol. 3467, 2005, pp. 185–203.

[47] J.B. Wells, A. Dimock, R. Muller, F. Turbak, A calculus with polymorphic and polyvariant flow types, Journal of Functional Programming 12 (3) (2002) 183–227.

[48] J.B. Wells, C. Haack, Branching types, in: Daniel Le Métayer (Ed.), ESOP (Programming Languages and Systems, 11th European Symposium on Programming, ESOP 2002, held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8–12, 2002, Proceedings), in: Lecture Notes in Computer Science, vol. 2305, Springer, 2002, pp. 115–132.